Introduction au Système d'Exploitation Unix/Linux

4^{ieme} partie: Utilitaires de fichiers Unix

B. Jacob

IC2/LIUM

11 octobre 2013

Plan

Plan

Pourquoi utiliser des commandes complexes ?

Pour manipuler des fichiers textes

- sed et awk sont des outils puissants mais complexes à utiliser
- les fonctions *Search and Replace* d'un éditeur de texte sont facile à utiliser mais fastidieuses si nombre d'opérations $\nearrow$

Dans quel cas les utiliser ?

Analogie au découpage de planches :

- pour une planche : utilisation d'une scie à main
  → mode d'emploi simple
  → temps de traitement court
  → on ne passe pas plus de temps à utiliser l'outil que d'apprendre à s'en servir
  (fonction *Search and Replace* d'un éditeur de texte)

- pour un lot de planches : utilisation d'une scie circulaire de menuisier
  → mode d'emploi plus complexe
  → temps de traitement long mais automatique
  → gain de temps par rapport au découpage du lot de planches avec une scie à main (`sed` et `awk`)

Plan

Présentation

`sed [-n] [-e` *script*`] [`*fichier*`]`
ou
`sed [-n] [-f` *fichier_script*`] [`*fichier*`]`

- `sed` : *streameditor*

  C'est un filtre, donc par défaut
- Prend ses données (des lignes)
  - dans l'entrée standard
  - dans un fichier avec [*fichier*]
- Affiche ses résultats sur
  - la sortie standard
  - ne les affiche pas si `-n`

Présentation

`sed` modifie les lignes à partir d'un `script`

- `script` = commandes d'édition
  - en ligne avec `-e` *script* (1 commande par `-e`)
  - contenu dans un fichier avec `-f` *fichier_script*

  Si il y a seulement une option `-e` et pas de `-f` alors on peut omettre le `-e`

Commandes d'édition

`[addresse [,adresse]] fonction [arguments]`

- sélectionne les lignes selon les `adresses`
- leur applique une `fonction` de sed avec ses `arguments`

Adresses de `sed`

- vide → toutes les lignes sont sélectionnées
- *n* → la ligne de numéro *n* dans chaque fichier
- $ → seulement la dernière ligne de chaque fichier
- *n1*,*n2* → n°de lignes entre *n1* et *n2*
- `/expression reguliere/` → définit un contexte d'adresse

Contexte d'Adresses

- décrit le contexte dans lequel doivent être les lignes sélectionnées.
- définit par une `/expression reguliere/`
  - `sed` supporte les expressions régulières étendues (voir cours 2)
  - + `\n` (NEWLINE)

Fonctions de sed

Il existe beaucoup de fonctions (Commandes d'édition) (`man sed`)
Parmis les plus utilisées :

- `a` (append) ajoute du texte
- `c` (change) remplace la ligne
- `d` (delete) efface la ligne
- `w` *fichier* (write) écrit la ligne dans *fichier*

Exemples

```
$ cat script.sed
1a\
zzzzzz

$ cat test_sed.txt
aaaaaa
bbbbbb
cccccc

$ sed -f script.sed test_sed.txt
aaaaaa
zzzzzz
bbbbbb
cccccc
```

Fonction s de sed

C'est **la** plus utilisée

`s/reg-exp/remplacement/flags`

- `s` (substitute)
- `reg-exp` expression régulière de sed
  - Stockage `\(` `reg-exp` `\)`
  - Rappel `\1` `\2` `\3` ...
- `flags` :
  - `g` (global) (faire toutes les substitutions de la ligne)
  - `n` avec $n \in [1-512]$ remplace seulement la n^{ieme} occurence
  - `p` (print) si ok sort la ligne sur la sortie standard
  - `w` *`fichier`* (write) si ok écrit la ligne dans *fichier*

Exemple simple

```
$ cat f
aaa bbb aaa bbb aaa
aaa ccc ddd

$ sed -e 's/aaa/AAA/' f
AAA bbb aaa bbb aaa
AAA ccc ddd

$ sed -e 's/aaa/AAA/g' f
AAA bbb AAA bbb AAA
AAA ccc ddd

$ sed -e 's/aaa/AAA/3' f
aaa bbb aaa bbb AAA
aaa ccc ddd
```

Exemple avec stockage et rappel des champs

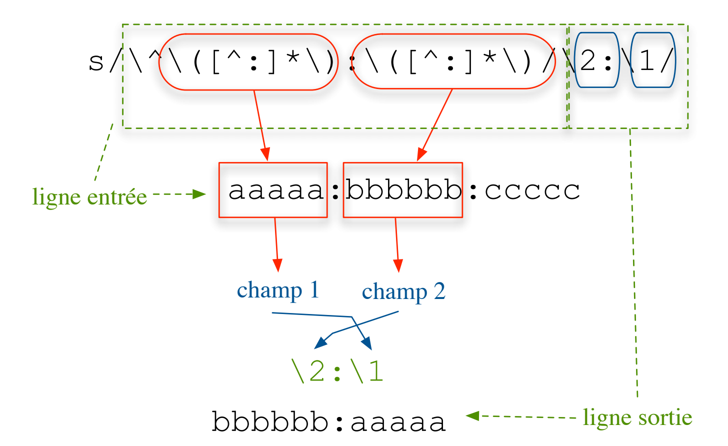
Fonction d'inversion des 2 premiers champs

```
$ cat f
aaaaa:bbbbbb:ccccc
ddddd:ee:ff
ggg:hhhhh:iiii

$sed -e 's/\^\([^:]*\):\([^:]*\)/\2:\1/' f
bbbbbb:aaaaa:ccccc
ee:ddddd:ff
hhhhh:ggg:iiii
```

Exemples avec stockage et du rappel des champs

Synopsis



Exercices

Exercices sur `sed`

Plan

Introduction

Format

```
awk -f programme fichier1 fichier2 ...
```

`awk` est un filtre. Donc, par défaut,

1. prend ses données (des fichiers)
    - dans l'entrée standard n°0 associée au clavier
    - dans des fichiers si `fichier1 fichier2...` sont spécifiés
2. leur applique le traitement `programme`
3. affiche ses résultats sur la sortie standard n°1 associée à l'écran

Programme

Un programme awk à la forme générale suivante

`BEGIN` { **instructions à faire avant de lire les données** }
sélecteur { **actions** }
...
`END` { **instructions à faire à la fin de la lecture les données** }

- Il peut y avoir plusieurs couples **sélecteur**/**actions**.
  Pour 1 couple, `awk`
  1. sélectionne les lignes des entrées avec le **sélecteur**
  2. exécute **actions** sur celles ci
- `BEGIN` actions exécutées au début du fichier.
- `END` actions exécutées à la fin du fichier.

Variables utlisateurs

Il suffit d'affecter un nom avec une valeur. Par exemple :

```
max = 2
nblignes = 0
```

Les tableaux sont associatifs (indexés par une chaînes de caractères). Par exemple :

```
couleurs[‘‘bleu’’] = 10
couleurs[‘‘blanc’’] = 20
couleurs[‘‘rouge’’] = 30
```

Accès aux valeurs des tableaux :

```
coul = ‘‘blanc’’
couleurs[coul]  --> désigne la valeur 20
```

Variables prédéfinies

Dans tous progammes Awk, les variables suivantes sont disponibles :

FS séparateur de champ en entrée.
- par défaut `FS = " "`
- si on veut traiter un fichier dont les champs sont séparés par des " :" alors il faudra mettre l'instruction `FS=":"` dans la section `BEGIN`

NR : nombre de lignes lues

RS : séparateur de lignes (par défaut `''\n''`)

LOGNAME : nom du fichier actuel

FNR : nombre d'enregistrements dans le fichier actuel

Champs

Les champs d'une ligne sont identifiés par des numéros.

Dans une expression awk

$0	représente la ligne d'entrée dans sa totalité
$1, $2, ...	représentent le premier, le deuxième ... champ de cette ligne

Par exemple, si " :" est le séparateur de champs alors

ligne d'entrée = aaaaa : bbbb : ccc : dd

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

champs = $1 $2 $3 $4

Instructions

Une instruction se compose de deux parties :

1. la sélection des lignes
2. les actions à faire sur ces lignes

Sélecteur

3 formats principaux à base d'expressions régulières (`regexp`)

- `regexp`
- `regexp debut , regexp fin`
- *champs* `~regexp` (ou `!~regexp`)

Les expression régulières de awk sont les expressions régulières étendues.

Sélecteur `regexp`

Sélectionne toutes les lignes pour lesquelles `regexp` est vraie.

Exemples :

- `NR==10` sélectionne la 10^{ieme} ligne
- `/coucou/` sélectionne toutes les lignes dans lesquelles il y a le mot "coucou"
- `NF > 4` sélectionne toutes les lignes qui ont plus de 4 champs

Sélecteur `regexp debut , regexp fin`

Sélectionne une section du fichier comprise entre

- la première ligne qui vérifie `regexp début` et
- la dernière ligne qui vérifie `regexp fin`

Exemples :

- `NR==10,NR==20` sélectionne de la 10^{ieme} à la 20^{ieme} ligne
- `/début/,/fin/` sélectionne une section qui commence à la ligne contenant le mot "début" et qui fini à la ligne contenant le mot "fin"

Sélecteur *champs* ∼ `regexp`

champs ∼ `regexp` : sélectionne les lignes dans lesquelles `champ` vérifie `regexp`

champs !∼ `regexp` : sélectionne les lignes dans lesquelles `champ` ne vérifie pas `regexp`

Exemples :

- `$1 ~/toto|TOTO/` sélectionne les lignes dont le 1er champ est égal à "toto" ou "TOTO"
- `$1 ~/[cC]oucou/` sélectionne les lignes dont le 1er champ est égal à "Coucou" ou "coucou"

Actions

Une action `awk` =
une ou plusieurs instructions séparées par des " ;" ou des "\n".

Une instruction =

- une affectation,
- une structure de contrôle
- un appel à une fonction standard

Structures de contrôle

- Sélection
  - `if ( condition ) instruction [ else instruction ]`
- Boucle non bornée
  - `while ( condition ) instruction`
  - `do instruction while ( condition )`
- Boucle bornée
  - `for ( expression ; condition ; expression ) instruction`
  - `for ( varaible in tableau ) instruction`

Fonctions standards

Quelques exemples (faire un `man awk` pour une liste exhaustive)

print	affiche sur la sortie standard
length(s)	retourne la longueur de s
match(s, re)	retourne la position de l'expression régulière `re` dans la chaîne `s`
split(s, a, fs)	découpe la chaîne `s` dans le tableau `a` avec le séparateur `fs`
sprintf(fmt, expr, expr,...)	idem que le sprintf du langage C.
substr(s, m, n)	retourne la sous-chaîne de `s` commençant par `m` et de longueur `n`
getline	saisie de la ligne $0 au clavier

Exercices

Exercices sur `awk`